**UNIVERSIDAD TECNOLÓGICA DE PANAMÁ**

**FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES**

**DEPARTAMENTO DE INGENIERÍA DE SOFTWARE**

**GUÍA DE LABORATORIO N° 9**

Asignatura: Mantenimiento y pruebas de software Fecha de asignación: 08 de noviembre de 2021

Facilitadora: Ing. Belén Bonilla M. Grupos: 1SF131, 1SF132

Estudiante: Irving Villarreal

1. **TÍTULO** **DE LA EXPERIENCIA**: Dobles de prueba con PHPUnit.
2. **TEMAS**

* Automatización de pruebas de software
* Pruebas unitarias
* Dobles de pruebas

1. **OBJETIVO(S):**
2. Conocer la estructura de desarrollo de una prueba unitaria en PHPUnit.
3. Aprender a utilizar el framework PHPUnit para crear dobles de prueba.
4. **METODOLOGÍA:**
5. Trabajar de manera individual.
6. Seguir el procedimiento indicado en esta guía.
7. Subir su archivo con los resultados al espacio de trabajo en Teams.
8. Actualizar su repositorio en GitHub.
9. **PROCEDIMIENTO O ENUNCIADO DE LA EXPERIENCIA:**

**Test Doubles (Dobles de prueba):**

Cuando estamos escribiendo una prueba en la cual no podemos (o elegimos no) usar un componente real del que se depende (DOC), podemos reemplazarlo por un doble de prueba. El doble de prueba no debe comportarse exactamente como el DOC real; él solamente debe proveer la misma API que la dependencia real para que así el SUT piense que es una dependencia real.

Los métodos createMock($type) y getMockBuilder($type) que provee PHPUnit se pueden usar en una prueba para generar automáticamente un objeto que actúa como un doble de prueba del objeto original especificado, de tipo interfaz o nombre de clase. Este objeto doble de prueba se puede usar en cada contexto donde un objeto del tipo original se espera o necesita.

Los **stubs**, “proporcionan respuestas predefinidas a ciertas llamadas durante las pruebas, sin responder a otra cosa para la que no hayan sido programados”, es decir, los stubs son configurados para que devuelvan valores que se ajusten a lo que la prueba unitaria quiere probar, por lo que se utilizan para verificar el estado de los objetos. Los **mocks**, “son objetos preprogramados con expectativas que conforman la especificación de lo que se espera que reciban las llamadas”, es decir, son objetos que se usan para probar que se realizan correctamente llamadas a otros métodos, por lo que se utilizan para verificar el comportamiento de los objetos.

Diagrama

Descripción generada automáticamente

1. Vamos a empezar con la creación de un stub. Para ello vamos a crear un nuevo método vacío en la clase Calculadora llamado capturarEntradasPermutacion.
2. Crear en la clase de prueba (CalculadoraTest) un método llamado testCapturarEntradasPermutacion que nos permitirá realizar un stub que simule que el método capturarEntradasPermutacion de la clase Calculadora devuelve un arreglo con los valores 5 y 3.

// Se crea el doble de prueba para la clase Calculadora, método 'capturarEntradasPermutacion'

        $stub = $this->createMock('Calculadora');

        $stub->method('capturarEntradasPermutacion')

            ->willReturn(array(5, 3));

        $this->assertSame(array(5, 3), $stub->capturarEntradasPermutacion());

1. Ahora vamos a crear un mock. En la clase Calculadora debemos agregar un método calcularPermutacion que recibe como parámetros n y r, que llama al método calcularFactorial (que es un método vacío que recibe el parámetro $numero) en dos ocasiones, y que nos permite calcular el valor de permutación:
2. Luego, en la clase CalculadoraTest creamos la prueba para el método calcularPermutacion empleando un mock (necesitamos simular que el objeto que estamos probando llama a otro método).

/\* Se crea un mock para la clase Calculadora.

         Solo se hace mock al método calcularFactorial\*/

        $mock = $this->getMockBuilder('Calculadora')

             ->onlyMethods(array('calcularFactorial'))

             ->getMock();

/\* Se configuran las expectativas para el método calcularFactorial

        se llamará dos veces y devolverá 120 y 6, en cada ocasión, respectivamente. \*/

$mock->expects($this->exactly(2))

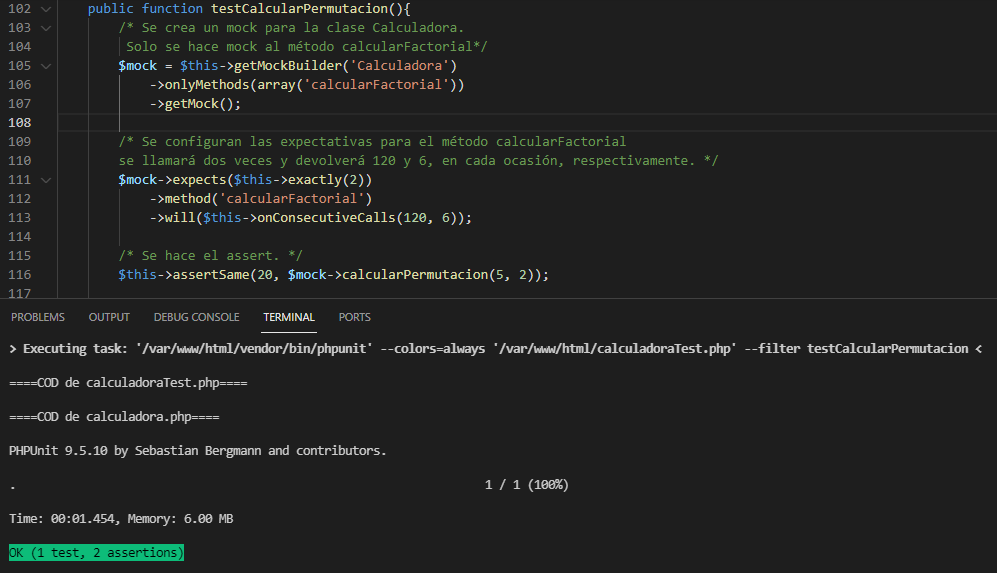
            ->method('calcularFactorial')

            ->will($this->onConsecutiveCalls(120, 6));

/\* Se hace el assert. \*/

$this->assertSame(20, $mock->calcularPermutacion(5, 2));

Corremos el método de prueba testCalcularPermutacion. Tome una captura de pantalla y colóquela aquí.

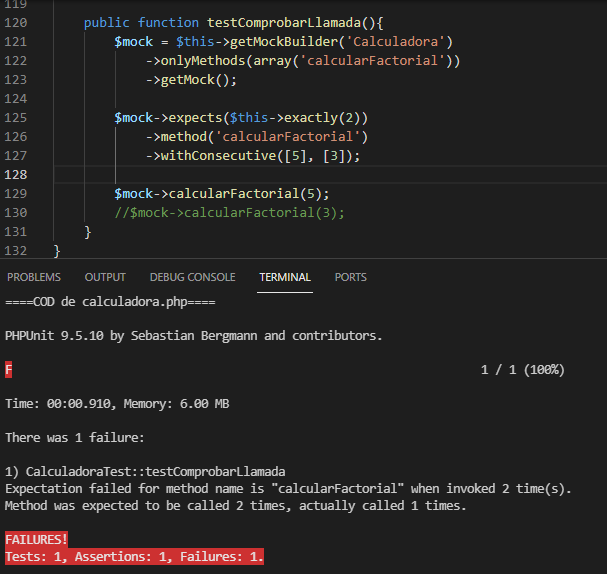


1. Vamos a realizar el mock agregando otros métodos para personalizar la construcción de este. Crear un método de prueba llamado testComprobarLlamada. Simular que método calcularFactorial ha sido llamado 2 veces y que se le ha enviado, el parámetro 5 la primera vez y el parámetro 3, la segunda vez.

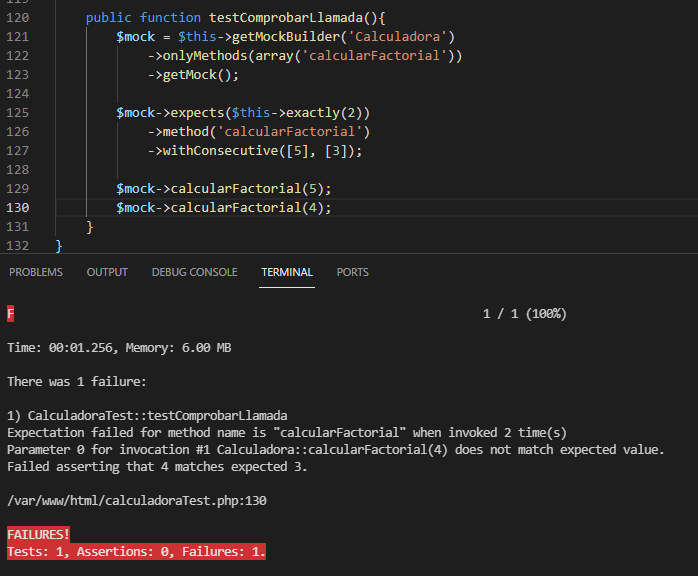
Corremos el método de prueba testComprobarLlamada. Tome una captura de pantalla y colóquela aquí.



Ahora, comente una de las llamadas a calcularFactorial. Vuelva a ejecutar la prueba unitaria. Tome una captura de pantalla del resultado obtenido y colóquelo aquí:

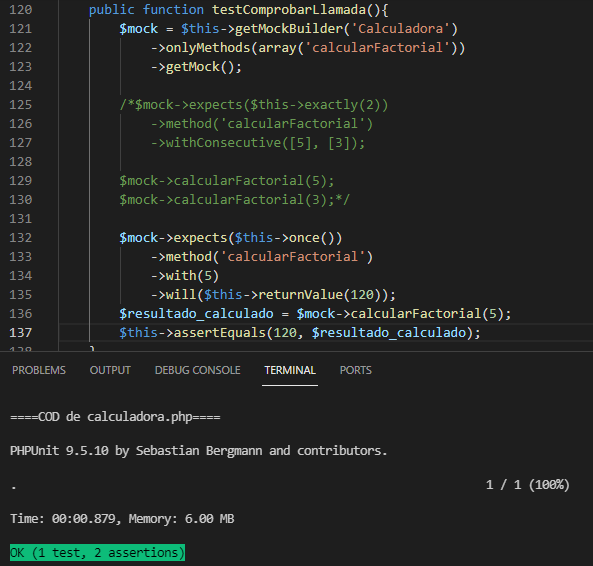


Quite el comentario de la llamada a calcularFactorial. Cambie el parámetro a la segunda llamada de calcularFactorial (cambie 3 por 4). Vuelva a ejecutar la prueba unitaria. Tome una captura de pantalla del resultado obtenido y colóquelo aquí:

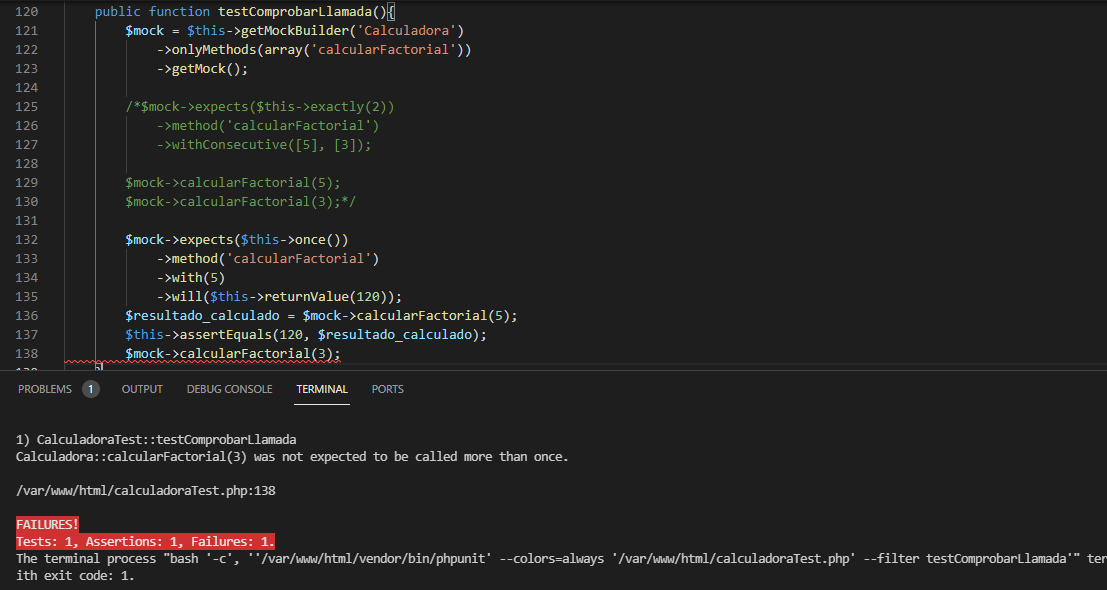


1. En este mismo método de prueba, simular que el método calcularFactorial ha sido llamado una vez con el parámetro 5 y que devuelve 120. Agregue una aserción assertEquals para comparar el resultado esperado con el calculado.

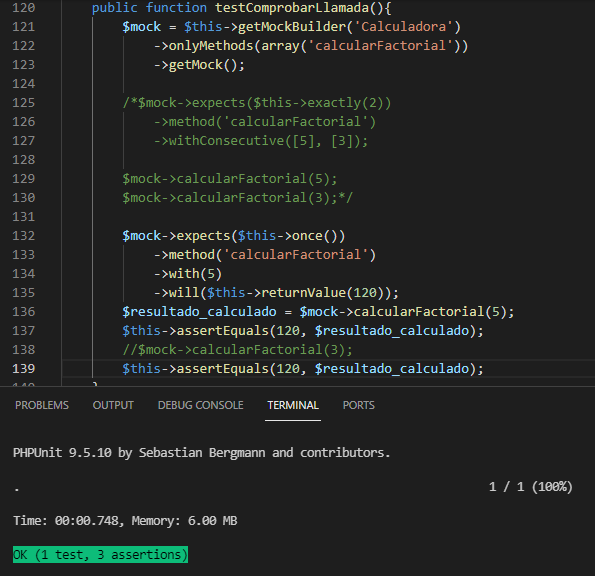
Corremos el método de prueba testComprobarLlamada. Tome una captura de pantalla y colóquela aquí.



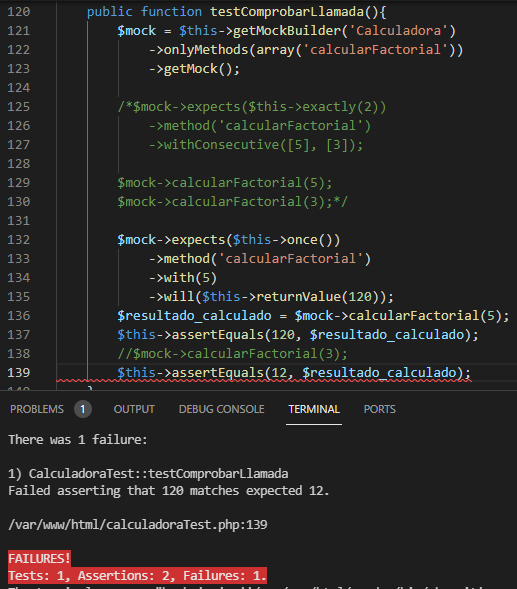
Ahora, agregue una llamada a calcularFactorial con el parámetro 3. Vuelva a ejecutar la prueba unitaria. Tome una captura de pantalla del resultado obtenido y colóquelo aquí:



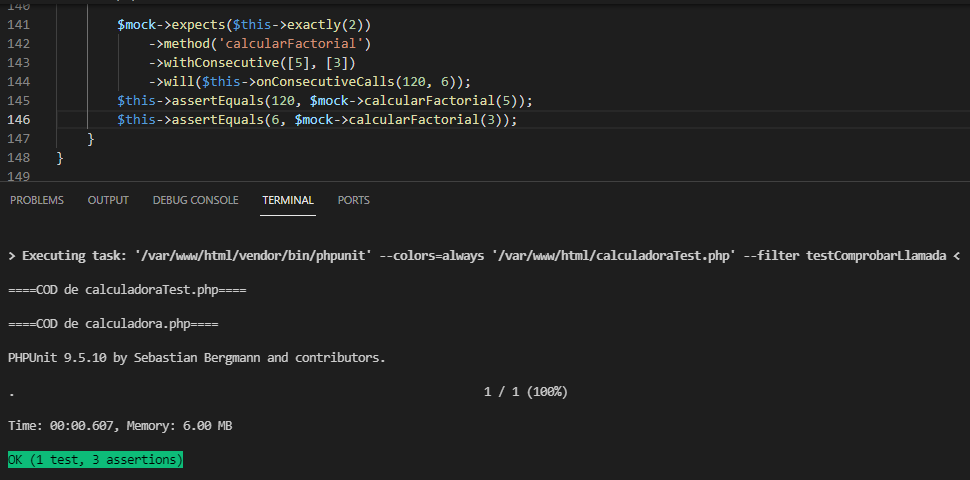
Quite la última llamada agregada. Agregue ahora un assertEquals para comparar el resultado esperado (120) con el calculado de manera simulada. Vuelva a ejecutar la prueba unitaria. Tome una captura de pantalla del resultado obtenido y colóquelo aquí:



Cambie el resultado esperado por otro número. Vuelva a ejecutar la prueba unitaria. Tome una captura de pantalla del resultado obtenido y colóquelo aquí:



1. En el mismo método de prueba, simular que el método calcularFactorial ha sido llamado 2 veces y que se le ha enviado, los valores 5 y 3 en cada llamada, respectivamente, y cada llamada ha devuelto respectivamente, los valores 120 y 6. Se hacen 2 assertEquals para comparar el resultado esperado vs. el calculado de manera simulada. Vuelva a ejecutar la prueba unitaria. Tome una captura de pantalla del resultado obtenido y colóquelo aquí:



1. Coloque aquí la url a su repositorio en GitHub: <https://github.com/irvingVillarreal/Calculadora_Irving.git>
2. **RECURSOS:** Computadora con: VSCode, XAMPP o un Docker con PHP, PHPUnit.
3. **RESULTADOS:** Proyecto PHP con el desarrollo realizado, laboratorio resuelto.